CHARACTERIZATION AND ADAPTIVE TEXTURE SYNTHESIS-BASED COMPRESSION SCHEME

Fabien Racape[1], Simon Lefort[1], Dominique Thoreau[1], Marie Babel[2] and Olivier Deforges[2]

1. Video Processing & Perception
Technicolor Research and Innovation
Cesson-Sevigne, France
fabien.racape@technicolor.com

2. INSA, IETR, UMR 6164
Universite Europeenne de Bretagne
Rennes, France

ABSTRACT

This paper presents an adaptive texture synthesis-based compression scheme, where textured regions are detected and removed at encoder side, allowing the decoder to use texture synthesis to fill them. The detection relies on locally adaptive resolution segmentation. According to results shown by synthesis algorithms, they need to be parameterized according to the patterns to be synthesized. In this framework, the synthesizer gets its parameters from DCT feature-based texture descriptors. An adaptive pixel-based algorithm is used, relying on the comparison between current pixel neighborhood and those in an atypically shaped sample. Different neighborhood sizes are considered to better catch texture patterns. The framework has been validated within an H.264/AVC video codec. Experimental results show significant bit-rate saving at similar visual quality.

1. INTRODUCTION

In state-of-the-art compression schemes, pixel-wise redundancy is reduced by using predictions and transformed domain operations. However, classical spatio-temporal approaches, exploiting redundancy based on the mean squared error (MSE) criterion, are not able to take visual redundancy into account. Detailed textures may seem nearly stationary for the Human Visual System, but totally irregular according to MSE criterion. At the same time, texture synthesis algorithms [1, 2, 3, 4] have shown promising results. The purpose of new coding schemes is to detect regions where exact positions of texture patterns are irrelevant for the human eye. The whole regions are not encoded since a few patterns are sufficient enough to synthesize satisfactory regions.

This paper presents a framework that can be integrated in standard coding schemes. One the main novelties of this scheme comes from the fact that texture algorithms have to be parameterized in order to give their best results. Thus, the scheme includes a texture characterization step based on DCT-domain descriptors that outputs the feature size of texture patterns to be synthesized. The remainder of the paper is organized as follows. In section 2 existing such schemes are presented. In section 3 is introduced the proposed framework. In sections 4 and 5 are described encoder and decoder designs in detail. Section 6 finally presents some experimental results.

2. RELATED WORK

One of the first synthesis-based compression scheme was presented in [5]. This approach includes an analysis in order to detect replaceable textures which are finally synthesized by a dedicated texture synthesis algorithm at decoder side. An interesting framework has been designed in [6] where some 8x8 blocks are removed at the encoder side and synthesized by the decoder. The segmentation first classifies blocks into structures, corresponding to boundaries of objects, and textures. The first category is classically encoded and the latter is removed and synthesized at decoder. The segmentation is based on simple edge detector thresholding. To avoid temporal inconsistencies, motion estimation is considered in selecting patches which will be used for texture synthesis. The latter is performed using the algorithm presented in [3], which appeared hard to exploit in our tests, since 8x8 blocks offer poor overlap for the Graphcut technique. The work presented in [7] proposes a closed-loop analysis-synthesis approach. As in [6], groups of pictures (GOP) are considered for a spatio-temporal scheme. Each potential region from a GOP is both analyzed and synthesized, using side information from texture analyzer. A first synthesizer is designed for *Rigid textures* with global motion, which has a great similarity to global motion compensation (GMC). Another synthesizer, inspired by the patch-based approach developed in [3], processes *Non-rigid textures* with local and global deformations. The scheme has recently been upgraded [8] with photometric corrections, using Poisson editing and covariant cloning. Another framework for image compression is proposed in [9] where both textural and structural blocks can be removed at encoder side. The textures are synthesized at decoder using [3], while a binary edge information helps an inpainting method to retrieve structural ones. To keep color coherency, a draught board of blocks is preserved, limiting the bit-rate saving. The problem of these gradated regions is tackled in [10] with parameter assistant inpainting.

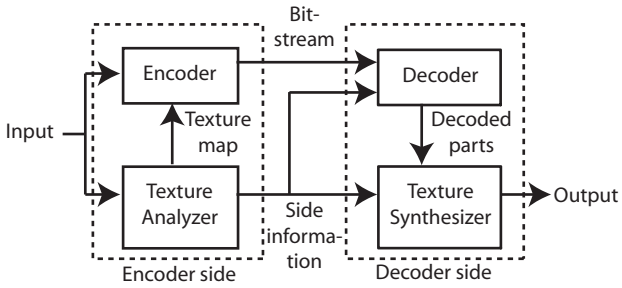


Figure 1: Framework overview.

However, several pixel-based synthesis algorithms from the literature [1, 2, 4] also provide promising visual performance for a large range of textures, until they get the right

parameters depending on the patterns to synthesize. Thus this work proposes to explore their efficiency in the compression context, by adding a coherent segmentation and a preprocessing step for texture synthesis that provides such parameters.

3. FRAMEWORK OVERVIEW

The proposed framework is depicted in figure 1. First, input images are analyzed to detect textured regions that can be synthesized by the algorithm used at decoder side. Thus this part of the framework is divided into two main steps: a segmentation step and a characterization step. A resulting texture map is sent to the encoder and side information, describing texture patterns, can potentially been sent. Then, pointed out regions are partially removed, whereas structural regions are classically encoded. Removed regions are synthesized from neighboring small surfaces of texture, also classically encoded. In the following, the patches denote these samples of texture which serve as inputs for synthesis. At the decoder side, the bit-stream and potential side information to locate encoded regions are both used to build and locate structural parts, whereas the labeled removed regions are synthesized using a new adaptive pixel-based algorithm. Like encoder side analysis, the synthesis contains a characterization step, which outputs required parameters for texture algorithm.

The next section describes more precisely the analysis done at the encoder side.

4. TEXTURE ANALYSIS

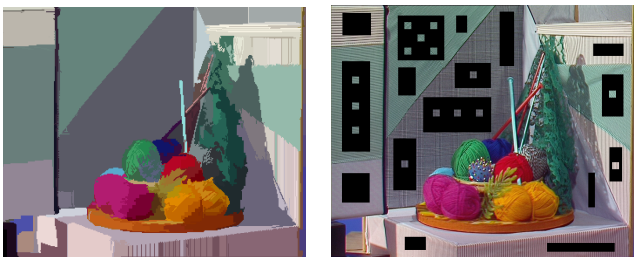
The goal is here to provide a complete and coherent texture analysis process, containing an adapted block-based segmentation and texture characterization.

4.1 LAR segmentation

Since the proposed framework requires a robust segmentation process that can be adapted to our context, the content-based codec LAR (Locally Adaptive Resolution) presented in [11] has been chosen for its region handling functionality. A spatial coder partitions images into variable-size blocks, relying on a homogeneity criterion based on a gradient thresholding. A quadtree topology is adopted in order to get a non-overlapping distribution of blocks. Then, a segmentation step merges adjacent blocks. This step uses a hierarchical approach and non-symmetrical distances taking into account the surface areas. The distance between two adjacent regions is the average between:

- Mean weighted distance, which is computed from mean Y values,
- Gradient distance, which corresponds to the local gradients along shared borders.

Two parameters steer the segmentation. First, the described distance is hierarchically thresholded. Then, a chromatic control involving the mean distance of chroma components between adjacent regions supervises the merge operation. Figure 2 gives, on the left side, an example on the *Wool* frame where regions are represented with their mean value. The following describes the adaptation of the LAR-region output map to the framework constraints.



LAR Regions | Removed texture regions

Figure 2: Adapted regions, black regions are removed to be synthesized at decoder side.

4.2 Adapting labels to texture synthesis compliant maps

In order to be compliant with block-based standard compression schemes, segmented regions are aligned on a block-based grid, i.e. 8x8 or 16x16 blocks. Each block containing several regions is considered as structure like in [6]. The others are considered as texture and will be removed by the encoder. Since the grid follows 16x16 macroblocks (MB), no additional information is required when using h.264 encoder. Binary side information may be sent in other configurations to locate structure blocks. Figure 2 shows such regions in the *Wool* frame with a 16x16 grid. Large textured regions require anchor preserved blocks in order to prevent artefacts. After experiments, it has been decided to encode some anchor macroblocks at strategic locations depicted in figure 2 since the region size exceeds a fix number of MBs in width or height. Coupled with a confidence-based synthesis order described in next section 5.2, they prevent visible seams at synthesis junctions. These segmented textures are then characterized to get input information about texture patterns for synthesis.

4.3 Texture characterization

This part of the scheme is used at both encoder and decoder sides. It outputs the parameters required by the texture synthesizer in order to give its best results. Thus, using characterization enables the encoder to decide whether the texture can be synthesized or not. Indeed, if the characterized texture requires impossible parameters, for example too large patterns, it is finally classically encoded. The main parameter for our pixel based algorithm corresponds to a size of neighborhood window. Thus the characterization step computes descriptors from different sizes. The descriptors are derived from those described in [12] which are computed from the Fourier transform. Since the characterization requires little sizes of blocks, i.e. inferior to 32 pixels wide, descriptors from DCT domain have been chosen. Indeed, the Fourier transform outputs descriptors with a lower resolution in frequency, which is a problem to describe a signal from a small window. Like in [12], descriptors are computed from concentric circles represented in figure 3, following:

$$D_{DCT}(\lambda) = \int_{\theta=0}^{\pi/2} |C(\lambda,\theta))|^2 \qquad (1)$$

where $C(\lambda,\theta)$ corresponds to the DCT coefficient at the location $(\lambda cos\theta, \lambda sin\theta)$, except they are integrated on a quarter of circle. The DC coefficient is not taken into account in

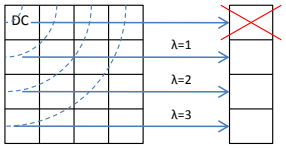


Figure 3: Descriptor computed on a 4x4 DCT block.

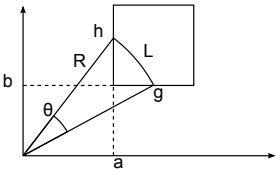


Figure 4: Descriptor Integral on a discrete array.

order to be invariant to changes of average luminance, so the descriptors vector has size of block minus one coefficient. A large set of blocks inside the texture region are randomly chosen to compute descriptors at different sizes centered at the same position.

Since the scheme uses discrete transform, figure 4 depicts the integral computation on a particular coefficient at position (i, j). The DCT value is weighted with the length L of the arc of a circle, crossing position (i, j) from a to b, given by $L = R * \theta$ where

$$\theta = arcsin\left(\sqrt{1-\frac{b^2}{R^2}}\right) - arcsin\left(\frac{a}{R}\right). \qquad (2)$$

The average descriptors are then computed for each size and analyzed. According to [1] and our experiments, the size of the compared neighborhood has to be greater than the elementary pattern of the texture, to produce a visually good result. The first coefficient of each block corresponds to a single variation of luminance over the block used for DCT computation. Experiments on a large set of texture patches show that if the first coefficient is greater than any others in the descriptors vector, the block size is smaller than an elementary pattern. Indeed, the first coefficient of each block corresponds to a single variation of luminance over the block used for DCT computation. The output size parameter corresponds to the minimum size NxN of DCT block that provides a non-monotone decreasing vector, which is given by

$$N_{out} = \min_{N}\left\{B_{DCT}^{N\times N} | D(1) \neq \max_{\lambda=[2;N-1]}\{D(\lambda)\}\right\} \qquad (3)$$

where B is the DCT block of size N. Thus, computing descriptors at random locations will serve to approximate the feature size of texture patterns. Typically, descriptors of sizes from 4x4 up to 16x16 pixels are computed for each location.

The next section describes the synthesis of the segmented regions, using parameters from texture characterization.

5. TEXTURE SYNTHESIS

The proposed synthesizer is based on the work presented in [1]. This algorithm is pixel-based since the output surface

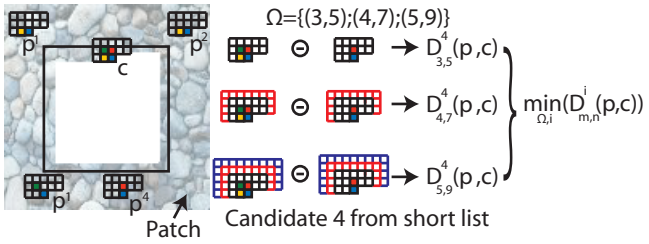


Figure 5: Finding best matching neighborhood.

is processed one pixel at a time. It relies on the matching between current pixel neighborhood and those in a texture patch. The criterion used for matching is the Sum of Square Error (SSE). Although this algorithm provides pleasing visual results for texture synthesis only, it has to be well-adapted to compression. First, it has been designed to rely on a given texture patch only, whereas our synthesis aims at exploiting all the available data (decoded blocks). Then, algorithm parameters such as synthesis order and neighborhood size and shape have to be reconsidered.

5.1 Adaptive neighborhood size using texture characterization

Texture characterization enables the synthesizer to get an approximate neighborhood. In order to refine the neighborhood sizes for matching, a set of neighborhood sizes are tested, for example 7x7 and 9x9 if the best descriptor is 8x8 large. The chosen distance minimizes the norm distance

$$D_{m,n}(p,c) = \frac{\sum_{k=0}^{N_{m,n}}(p_k - c_k)^2}{N_{m,n}} \qquad (4)$$

where $N_{m,n}$ is the number of pixels contained in the neighborhood of current pixel, m and n respectively representing its height and width, c_k and p_k denote the kth pixel's luminance in the current neighborhood and the considered one in the patch respectively. Considering various neighborhood sizes enables to better catch texture patterns size and shape. This process is illustrated in figure 5 where three neighborhood sizes $\{(3;5)(4;7)(5;9)\}$ are competing.

Experiments show that a synthesis with a lot of changes in neighborhood sizes does not give visually good results. In order to avoid this kind of issue, we propose to favor the neighborhood sizes that have been chosen by a majority of previously synthesized pixels, by mean of a global weight.

5.2 Synthesis scan order

The goal is here to exploit available data which are, at this point, the only confident data. The latter corresponds to surrounding pixels, which are whether previously decoded or synthesized. Thus, raster scan order is clearly not adapted. As in [9], a scan order depending on a synthesis-coherent confidence map is adopted. The map building law is depicted in figure 6, where initial computing from previously decoded pixels is represented on the left side and the map update during synthesis on the right side. First, previously decoded pixels are initialized with a unique confidence value. In order to be coherent with the synthesis, the neighborhood size used for texture synthesis serves for computing the confidence of available pixels. Thus, this confidence value ω is computed

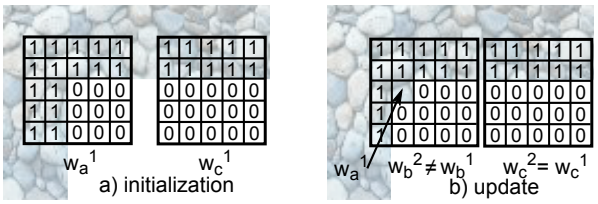


Figure 6: Confidence map order.

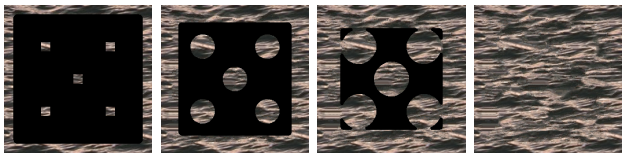


Figure 7: Pixel-based synthesis order.

as follows:

$$\omega(i,j) = \sum_{k=-N/2}^{N/2} \sum_{l=-N/2}^{N/2} \omega(i+k, j+l). \qquad (5)$$

After having synthesized the most confident pixel, the map is updated by assigning the previously computed $\omega(i,j)$, which is shown in figure 6 b).

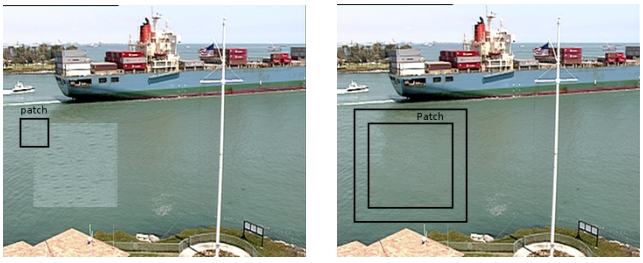
5.3 Texture patch design

Texture synthesizers from the literature process with a rectangular patch. Its relevance is essential to ensure an efficient synthesis in our context. In the literature, patch choices are usually either not described or correspond to cropped proximate blocks [6]. However, when removing large regions, one can note that cropping a patch closed to the texture region to be synthesized can lead to inconsistencies between candidate pixels and previously reconstructed borders. For instance, figure 8 a) shows a selected region in which there is a variation of luminance, leading to a failed synthesis. That is, the patch contains the texture around the region to be synthesized, which is depicted in figure 8 b). The patch has to be large enough, according to the size of neighborhood. A compromise is done at encoder side, if texture region are too small and patterns too large, then the region is not classified as texture. Reversely, large texture regions require anchors in order to prevent pixel-based synthesis to fail, so we decide to keep some MBs since the size exceeds a fix width or height. Figure 7 depicts the use of such anchors, following the confidence-based scan order.

5.4 Neighborhood optimization

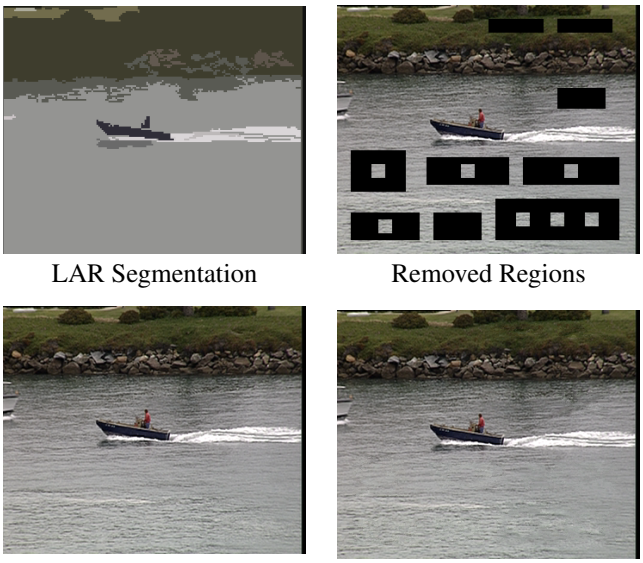
In order to improve the research process, we propose to use the exact neighborhood matching described in [13]. A shortlist of candidate neighborhoods in the patch is first computed for the current location. It corresponds to the set of neighborhoods in the patch containing at least one direct neighbor which exactly matches respective current pixel neighbor. Then the SSE distance is computed for the all neighborhoods in the shortlist and the best candidate is chosen like in [1]. The only change lies in the scan order which defines the causal direct neighbors. This technique enables a great adaptation to our context in which high frequency discontinuities must be avoided.

Some results are provided in the next section, in particular visual comparisons between the persented pixel-based synthesis and a patch-based method.



a) square patch b) chosen patch

Figure 8: Texture patch design.



LAR Segmentation Removed Regions

Decoded I frame with QP=20 Final result

Figure 9: Visual Results on the *Coastguard* CIF sequence.

6. EXPERIMENTAL RESULTS

The framework has been implemented within the JM joint model [14]. Presented results are given for intra frames sequences *Wool, Container, Coastguard* and *Soccer*. A set of four Quantization Parameters (QP)$\{15,20,25,30\}$ is considered. The segmentation is aligned on a 16x16 grid. The retained patch has been set to 16 pixels of width and three sizes of neighborhoods $\{(4,7);(5,9);(6,11)\}$ are competing. Table 1 shows the bit-rate saving at similar visual quality for the different sequences and QP. One can see the promising performance, i.e. the scheme achieves up to 20% bit-rate saving on *Soccer* and 14% on *Container*. One notes the other good results on *Coastguard* containing a large part of water texture. Figure 9 shows the visual results for the *Coastguard* sequences encoded with $QP = 20$, where 22.2% of the frame has been removed and synthesized. Figure 10 aims at comparing this pixel-based scheme and the patch-based from [3] on the *coastguard* and *Morocco Trial* sequences.

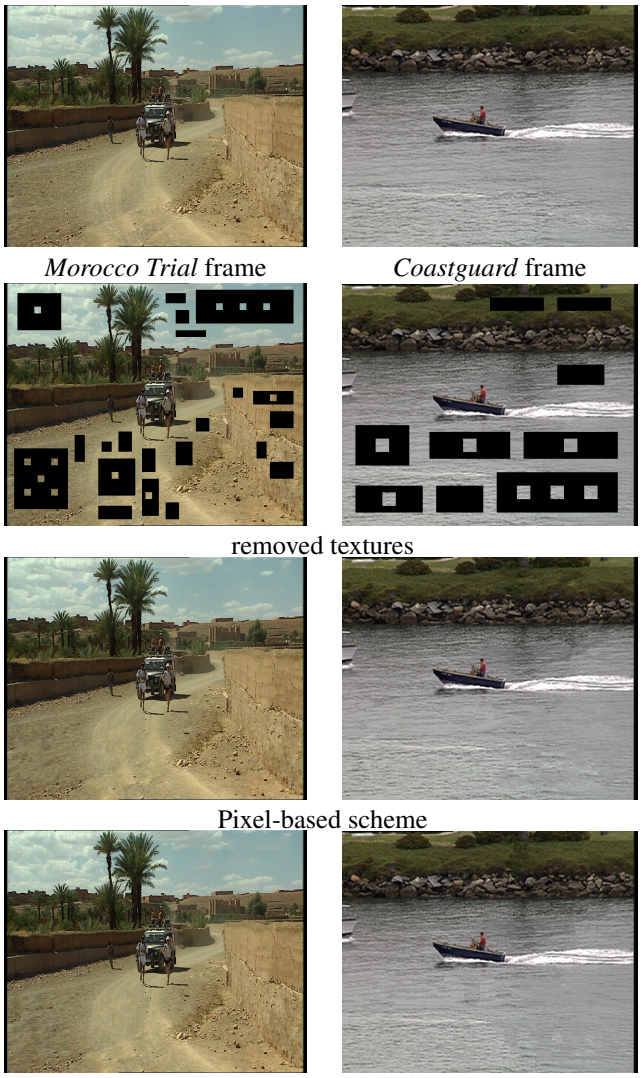


Figure 10: Comparison between pixel-based and patch-based synthesis.

One can note comparable results for most of textures, for example with the trodden earth in the *Morocco Trial* sequence. However, results on water in the *Coastguard* sequence points out that the pixel-based technique avoids edge artefacts for smooth textures.

7. CONCLUSION AND FUTURE WORK

A framework based on a new adaptive texture synthesis has been presented. The synthesis is pixel-based since it relies on the comparison between current pixel neighborhoods and those in an atypically shaped patch. In order to better catch texture patterns, a texture characterization is carried out to parameterize the synthesizer, in order to choose the best candidate among different neighborhood sizes and shapes. The framework has been validated within an H.264/AVC video codec. Experimental results show significant bit-rate saving compared to H.264/AVC. Ongoing researches focus on the characterization of detected textures to improve synthesizer adaptation as well as combining patch-based and pixel-based approaches.

QP	15	20	25	30
Sequences	Bit-rate saving (%)			
Wool SD	11.3	9.6	7.9	6.2
Container CIF	14.7	11.7	8.4	3.4
Coastguard CIF	35.1	33.3	31.1	27.0
Soccer 4CIF	20.8	20.9	20.8	20.6

Table 1: Resulting bit-rate saving.

REFERENCES

[1] L.-Y. Wei and M. Levoy, "Fast texture synthesis using tree-structured vector quantization," in *Proceedings of ACM SIGGRAPH*, New York, USA, 2000, pp. 479–488.

[2] M. Ashikhmin, "Synthesizing natural textures," in *Proceedings of ACM Symposium on Interactive 3D Graphics*, 2001, pp. 217–226.

[3] V. Kwatra et al., "Graphcut textures: image and video texture synthesis using graph cuts," in *Proceedings of ACM SIGGRAPH*, 2003, pp. 277–286.

[4] V. Kwatra, I. Essa Aaron, and B. N. Kwatra, "Texture Optimization for Example-based Synthesis," in *Proceedings of ACM SIGGRAPH*, 2005, pp. 795 – 802.

[5] A. Dumitras, BG Haskell, A.C. Inc, and CA Cupertino, "A texture replacement method at the encoder for bit-rate reduction of compressed video," *IEEE TCSVT*, vol. 13, no. 2, pp. 163–175, 2003.

[6] C. Zhu et al., "Video coding with spatio-temporal texture synthesis," in *IEEE ICME*, 2007, pp. 112–115.

[7] P. Ndjiki-Nya, T. Hinz, and T. Wiegand, "Generic and robust video coding with texture analysis and synthesis," in *2007 IEEE ICME*, 2007, pp. 1447–1450.

[8] P. Ndjiki-Nya, D. Doshkov, and M. Koppel, "Optimization of video synthesis by means of cost-guided multimodal photometric correction," 2009.

[9] Dong Liu, Xiaoyan Sun, Feng Wu, Shipeng Li, and Ya-Qin Zhang, "Image compression with edge-based inpainting," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 17, no. 10, pp. 1273 –1287, 2007.

[10] Zhiwei Xiong, Xiaoyan Sun, and Feng Wu, "Block-based image compression with parameter-assistant inpainting," *Image Processing, IEEE Transactions on*, vol. 19, no. 6, pp. 1651 –1657, 2010.

[11] O. Deforges, M. Babel, L. Bedat, and J. Ronsin, "Color LAR codec: a color image representation and compression scheme based in local resolution adjustment and self-extracting region representation," *IEEE TCSVT*, vol. 17, no. 8, pp. 974–987, 2007.

[12] F. Smach, C. Lemaître, J.P. Gauthier, J. Miteran, and M. Atri, "Generalized Fourier descriptors with applications to objects recognition in SVM context," *Journal of Mathematical Imaging and Vision*, vol. 30, no. 1, pp. 43–71, 2008.

[13] M. Sabha, P. Peers, and P. Dutre, "Texture synthesis using exact neighborhood matching," *Computer Graphics Forum*, vol. 26, no. 2, pp. 131–142, 2007.

[14] "JM reference software version 14.0," *http://iphome.hhi.de/suehring/tml/download/*.